ГЛАВА 14

Модули

В главе 13 был представлен глобальный класс приложения и описана его роль в процессе обработки запросов ASP.NET Framework. Мы показали, как выглядят жизненные циклы приложения и запросов страниц, и объяснили роли *модулей* и *обработчиков*. Настоящая глава целиком посвящена модулям — мы рассмотрим, как они работают, каким образом создавать и пользоваться специальными реализациями и как управлять модулями, встроенными в ASP.NET. Мы также завершим описание различных видов методов, определенных в глобальном классе приложения — то, что было начато в главе 13.

К концу главы вы будете лучше понимать способ обработки запросов средой ASP.NET Framework и знать, как реализованы некоторые ключевые элементы функциональности.

Подготовка примера приложения

В этой главе мы собираемся продолжить пользоваться проектом `Events`, который был начат в главе 13. Данный проект содержит класс `EventCollection`, который позволяет записывать информацию о получаемых событиях жизненного цикла, и веб-форму `Default.aspx`, отображающую эту информацию с применением элемента управления `Repeater`.

В качестве напоминания в листинге 14.1 приведен код глобального класса приложения, на котором завершилась глава 13. Мы создали декларативные обработчики для событий жизненного цикла приложения и четыре обработчика для событий жизненного цикла запросов.

Листинг 14.1. Содержимое файла `Global.asax.cs` из проекта `Events`

```
using System;
using System.Web;

namespace Events {
    public class Global : System.Web.HttpApplication {
        private DateTime startTime;
        protected void Application_Start(object sender, EventArgs e) {
            EventCollection.Add(EventSource.Application, "Start");
            Application["message"] = "Application Events";
        }
        protected void Application_End(object sender, EventArgs e) {
            EventCollection.Add(EventSource.Application, "End");
        }
        protected void Application_BeginRequest(object sender, EventArgs e) {
            startTime = Context.Timestamp;
        }
```

```
        protected void Application_EndRequest(object sender, EventArgs e) {
            double elapsed = DateTime.Now.Subtract(startTime).TotalMilliseconds;
            System.Diagnostics.Debug.WriteLine(
                string.Format("Duration: {0} {1}ms", Request.RawUrl, elapsed));
        }

        protected void Application_PostAuthenticateRequest(object sender,
    EventArgs e) {
            if (Request.Url.LocalPath == "/Params.aspx" &&
                    !User.Identity.IsAuthenticated) {
                Context.AddError(new UnauthorizedAccessException());
            }
        }

        protected void Application_LogRequest(object sender, EventArgs e) {
            System.Diagnostics.Debug.WriteLine(
                string.Format("Request for {0} - code {1}",
                    Request.RawUrl, Response.StatusCode));
        }
    }
}
```

Обработчики событий жизненного цикла запросов использовались для решения трех задач: измерение времени обработки запроса, предотвращение просмотра веб-формы `Params.aspx` неаутентифицированными пользователями и запись в журнал информации о запросе.

Итак, у нас имеется полезная функциональность, однако она не особенно хорошо структурирована. В одном коде происходят три разных действия, но не сразу понятно, как различные операторы связаны друг с другом — это часто приводит к проблемам при внесении изменений в будущем. В реальном проекте код глобального класса приложения очень быстро может стать нечитабельным и, в конце концов, придется вырезать и вставлять код из файла `Global.asax.cs` в новый проект, если требуется его повторное использование.

Мы можем решить указанные проблемы, разбив код на *модули* — автономные единицы кода, которые могут реагировать на события жизненного цикла запросов, определенные в классе `HttpApplication`. Прежде чем делать это, необходимо удалить функциональность из глобального класса приложения, чтобы она не дублировала функциональность, которую планируется поместить в модули. Модификации кода глобального класса приложения можно видеть в листинге 14.2.

Листинг 14.2. Удаление функциональности из глобального класса приложения

```
using System;
using System.Web;

namespace Events {
    public class Global : System.Web.HttpApplication {

        protected void Application_Start(object sender, EventArgs e) {
            EventCollection.Add(EventSource.Application, "Start");
            Application["message"] = "Application Events";
        }

        protected void Application_End(object sender, EventArgs e) {
            EventCollection.Add(EventSource.Application, "End");
        }
    }
}
```

Как видите, удалены все обработчики для событий жизненного цикла запросов. В последующих разделах мы воссоздадим эту функциональность в наборе модулей.

Понятие модуля

Модули реализуют интерфейс `System.Web.IHttpModule`, в котором определены два метода, описанные в табл. 14.1.

Таблица 14.1. Методы, определенные в интерфейсе `IHttpModule`

Метод	Описание
`Init(app)`	Этот метод вызывается при создании экземпляра класса модуля и ему передается экземпляр `HttpApplication`. Применяйте данный метод для регистрации методов обработки для событий `HttpApplication` и для инициализации требуемых ресурсов
`Dispose()`	Этот метод вызывается при завершении обработки запроса. Применяйте данный метод для освобождения ресурсов, требующих явного управления

В приведенных далее разделах мы создадим набор модулей и покажем, как их регистрировать в ASP.NET Framework, чтобы они принимали участие в обработке запросов.

Время жизни модуля

Экземпляры модулей создаются во время создания нового объекта `HttpApplication`. Каждый объект `HttpApplication` получает собственный набор объектов модулей. При создании экземпляра модуля вызывается метод `Init()` и, подобно объекту `HttpApplication`, с которым модуль ассоциирован, модуль может использоваться для обработки множества запросов (хотя и по одному запросу за раз). При написании кода модуля следует помнить, что в любой момент времени может существовать несколько экземпляров модуля, и позаботиться об отсутствии непредусмотренных последствий обработки множества запросов (так, например, удостоверьтесь, что сбрасываете состояние модуля при получении события `BeginRequest`, а не в методе `Init()`).

В классе `HttpApplication` также определен метод `Init()`, который вызывается, когда был вызван метод `Init()` для всех созданных объектов модулей. Этот метод можно применять для регистрации обработчиков событий, определяемых модулями, что будет демонстрироваться далее в этой главе.

Создание модуля

Мы собираемся начать с создания модуля, который предотвращает просмотр веб-формы `Params.aspx` неаутентифицированными пользователями. Для этого мы добавили в пример проекта новый элемент по имени `ParamsModule.cs`, используя шаблон элемента ASP.NET Module (Модуль ASP.NET). Модули — это просто классы C#, и в листинге 14.3 показано содержимое файла `ParamsModule.cs`, сгенерированное Visual Studio.

Листинг 14.3. Начальное содержимое файла `ParamsModule.cs`

```
using System;
using System.Web;

namespace Events {
    public class ParamsModule : IHttpModule {
```

```
        public void Init(HttpApplication context) {
            context.LogRequest += new EventHandler(OnLogRequest);
        }

        public void Dispose() {
        }

        public void OnLogRequest(Object source, EventArgs e) {
        }
    }
}
```

Мы удалили ряд комментариев и привели код к удобному для чтения виду, чтобы легче было понять, что было создано средой Visual Studio. В нашем распоряжении имеется класс по имени `ParamsModule`, который реализует интерфейс `IHttpModule` и регистрирует пустой метод обработчика для события `HttpApplication.LogRequest`. При такой отправной точке реализовать функциональность, требуемую для нашего простого модуля защиты, довольно просто. Внесенные в код изменения показаны в листинге 14.4.

Листинг 14.4. Реализация функциональности класса `ParamsModule`

```
using System;
using System.Web;

namespace Events {
    public class ParamsModule : IHttpModule {

        public void Init(HttpApplication app) {

            app.PostAuthenticateRequest += (src, args) => {
                if (app.Request.Url.LocalPath == "/Params.aspx" &&
                        !app.User.Identity.IsAuthenticated) {
                    app.Context.AddError(new UnauthorizedAccessException());
                }
            };
        }

        public void Dispose() {
        }
    }
}
```

Для создания обработчика события `PostAuthenticateRequest` применяется лямбда-выражение. Обратите внимание, что мы должны обращаться к контекстным объектам через экземпляр `HttpApplication`, который передается методу `Init()`. (Лямбда-выражение использовалось лишь ради разнообразия. Мы считаем, что такой код легче читать в случае простых обработчиков событий вроде показанного, но вы можете применять обычные методы, как будет демонстрироваться в следующем примере.)

Совет. Обратите внимание на изменение имени параметра, передаваемого методу `Init()`. Нам нравится разумная согласованность в именовании переменных для контекстных объектов, так что экземпляры `HttpApplication` обычно получают имя `app`, объекты `HttpContext` — имя `context`, а объекты `HttpRequest` и `HttpReponse` — соответственно, имена `request` и `response`.

Регистрация модуля

Среда ASP.NET Framework не поддерживает автоматическое обнаружение классов модулей, поэтому мы должны предоставить среде детальные сведения о классе, чтобы он стал частью жизненного цикла. Это делается посредством файла `Web.config`; добавленные элементы можно видеть в листинге 14.5.

Листинг 14.5. Регистрация модуля в файле `Web.config`

```
<?xml version="1.0"?>

<configuration>
    <system.web>
      <compilation debug="true" targetFramework="4.5" />
      <httpRuntime targetFramework="4.5" />
    </system.web>

  <system.webServer>
    <modules>
      <add name="ParamsProtection" type="Events.ParamsModule"/>
    </modules>
  </system.webServer>

</configuration>
```

Совет. В листинге 14.5 показаны регистрационные элементы, которые будут работать с наиболее распространенной конфигурацией IIS и IIS Express. Регистрация модуля с другими конфигурациями IIS описана по адресу `http://msdn.microsoft.com/ru-ru/library/46c5ddfy(v=vs.100).aspx`.

Ответственность за обработку запроса возложена на сервер приложений, который конфигурируется с применением элемента `system.webServer`. Элемент `modules` содержит директиву для управления классами модулей, и с помощью элемента `add` производится регистрация нашего модуля. Элемент `add` внутри `modules` имеет атрибут `name`, определяющий имя, по которому можно ссылаться на модуль, и атрибут `type`, в котором указывается полностью определенное имя класса модуля. В листинге 14.5 в качестве имени для класса `ParamsModule` было задано `ParamsProtection`.

Совет. Можно также воспользоваться элементом `clear`, чтобы удалить все встроенные модули, и элементом `remove` для удаления отдельных модулей. Такая разновидность конфигурации рассматривается в главе 27.

Для тестирования модуля необходимо запустить приложение и запросить веб-форму `Params.aspx`. Появится то же самое сообщение об ошибке, которое отображалось при выполнении проверки авторизации в глобальном классе приложения.

Создание проекта для модуля

Защита конкретной веб-формы (хотя и слабая) — это то, что специфично для одиночного приложения. Другая функциональность, связанная с подсчетом времени обработки запросов и занесением сведений о них в журнал, является более общей и может применяться во многих проектах. В этом разделе мы создадим модули в отдельном проекте, который можно упаковывать и многократно использовать. Это позволит продемонстрировать ключевое преимущество модулей и важную функциональность, предназначенную для их регистрации.

Создание проекта Visual Studio

Для начала создадим новый проект под названием `CommonModules`. Для простоты мы добавим еще один проект в решение, созданное Visual Studio для проекта `Events`. (Решение — это контейнер для одного или нескольких проектов, и Visual Studio обычно создает решения по умолчанию; каждый проект является автономным, но решения позволяют работать с ними одновременно.)

Выберите пункт Add⇨New Project (Добавить⇨Новый проект) в меню File (Файл) среды Visual Studio и откроется диалоговое окно New Project (Новый проект) с обычным набором шаблонов проектов. Модули представляют собой простые классы C#, и они создаются с применением шаблона Class Library (Библиотека классов), который находится в категории Installed⇨Templates⇨Visual C#⇨Windows (Установленные⇨Шаблоны⇨Visual C#⇨Windows). Выберите указанный шаблон, введите `CommonModules` в поле Name (Имя) и щелкните на кнопке OK, чтобы создать новый проект.

Совет. В случае если запущен отладчик, добавить новый проект не удастся — пункты меню окажутся недоступными. Остановите отладчик, после чего пункты меню снова появятся.

В проект `CommonModules` понадобится добавить ссылку на сборку `System.Web`, чтобы получить доступ к интерфейсу `IHttpModule` и контекстным объектам. Щелкните правой кнопкой мыши на записи `CommonModules` в окне Solution Explorer (Проводник решения) и выберите в контекстном меню пункт Add Reference (Добавить ссылку). В открывшемся диалоговом окне Reference Manager (Диспетчер ссылок) отыщите сборку `System.Web` (она находится в разделе Assemblies⇨Framework (Сборки⇨Инфраструктура)) и отметьте флажок рядом с ней, как показано на рис. 14.1.

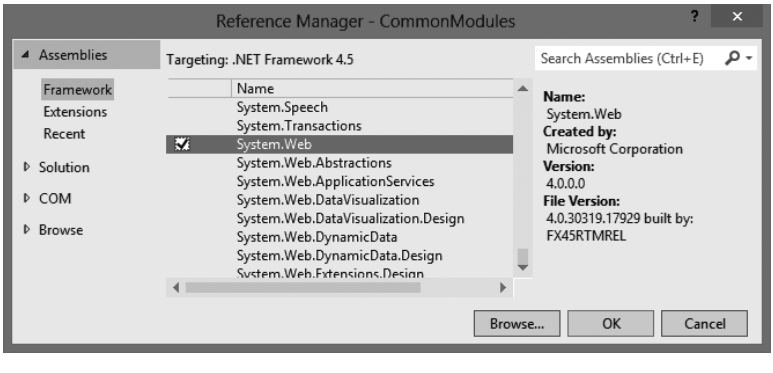

Рис. 14.1. Добавление ссылки на сборку `System.Web`

Щелкните на кнопке OK, чтобы закрыть диалоговое окно и добавить в проект ссылку на сборку `System.Web`.

Теперь, когда в решении присутствуют два проекта, необходимо сообщить Visual Studio, какой из них должен запускаться при запуске отладчика. Щелкните правой кнопкой мыши на проекте `Events` в окне Solution Explorer (Проводник решения) и выберите в контекстном меню пункт Set as Startup Project (Установить в качестве стартового проекта).

Среда Visual Studio добавляет в новые проекты библиотек классов файл класса по имени `Class1.cs`. Мы не собираемся пользоваться этим файлом, поэтому щелкните правой кнопкой мыши на записи `Class1.cs` в окне Solution Explorer и выберите в контекстном меню пункт Delete (Удалить).

Создание модулей

Мы будем создавать каждый модуль в своем файле класса. Щелкните правой кнопкой мыши на проекте CommonModules в окне Solution Explorer и выберите в контекстном меню пункт Add⇨Class (Добавить⇨Класс). Укажите в качестве имени LogModule.cs, щелкните на кнопке Add (Добавить), чтобы создать файл, и приведите его содержимое к виду, показанному в листинге 14.6.

Листинг 14.6. Создание класса модуля в файле LogModule.cs

```
using System;
using System.Web;

namespace CommonModules {
   public class LogModule : IHttpModule {

      public void Init(HttpApplication app) {
         app.LogRequest += HandleEvent;
      }

      public void Dispose() {
         // Ничего не делать.
      }

      protected void HandleEvent(object src, EventArgs args) {
         HttpApplication app = src as HttpApplication;
         System.Diagnostics.Debug.WriteLine(
            string.Format("Request for {0} - code {1}",
               app.Request.RawUrl, app.Response.StatusCode));
      }
   }
}
```

Мы определили класс, который реализует интерфейс IHttpModule и с помощью метода Init() регистрирует метод обработчика для события LogRequest. Этот метод обработчика события содержит тот же код, что применялся в главе 13, и выводит подробные сведения об обрабатываемых запросах в окно Output (Вывод) среды Visual Studio. Для последнего модуля мы добавили в проект CommonModules файл класса по имени TimerModule.cs. Содержимое этого файла представлено в листинге 14.7.

Листинг 14.7. Содержимое файла TimerModule.cs

```
using System;
using System.Web;

namespace CommonModules {
   public class TimerModule : IHttpModule {
      private DateTime startTime;

      public void Init(HttpApplication app) {
         app.BeginRequest += HandleEvent;
         app.EndRequest += HandleEvent;
      }

      private void HandleEvent(object src, EventArgs args) {
         HttpApplication app = src as HttpApplication;
         switch (app.Context.CurrentNotification) {
            case RequestNotification.BeginRequest:
               startTime = app.Context.Timestamp;
               break;
```

```
                case RequestNotification.EndRequest:
                    double elapsed =
                        DateTime.Now.Subtract(startTime).TotalMilliseconds;
                    System.Diagnostics.Debug.WriteLine(
                        string.Format("Duration: {0} {1}ms",
                            app.Request.RawUrl, elapsed));
                    break;
            }
        }
        public void Dispose() {
            // Ничего не делать.
        }
    }
}
```

Этот модуль определяет, сколько времени заняла обработка запроса, за счет использования свойства `HttpContext.TimeStamp` и обработки событий `BeginRequest` и `EndRequest` — в точности, как это делалось, когда данная функциональность находилась в глобальном классе приложения. В рассматриваемом примере мы применяем свойство `HttpContext.CurrentNotification` для демонстрации того, что модули могут использовать такой же подход к обработке множества событий, как и показанный в главе 13.

Регистрация модулей

Мы могли бы зарегистрировать созданные модули в файле `Web.config` проекта `Events`, но нам не нравится такой подход — мы предпочитаем иметь автономные единицы функциональности, к тому же идея добавления элементов в файл `Web.config` одного проекта, чтобы применять классы из другого проекта, нам представляется неудачной.

Вместо этого мы воспользуется приемом, который позволит модулям регистрировать себя в ASP.NET Framework автоматически без необходимости в конфигурационных записях. Начиная с версии 4, в среде ASP.NET Framework поддерживается возможность указания кода, который должен быть выполнен прямо перед вызовом метода `Application_Start()` глобального класса приложения.

Нам нужно создать класс, содержащий операторы, которые требуется выполнить. С этой целью мы добавляем к проекту `CommonModules` файл класса под названием `ModuleRegistration.cs` с содержимым, представленным в листинге 14.8.

Листинг 14.8. Содержимое файла `ModuleRegistration.cs` из проекта `CommonModules`

```
using System;
using System.Web;
[assembly: PreApplicationStartMethod(
    typeof(CommonModules.ModuleRegistration), "RegisterModules")]
namespace CommonModules {
    public class ModuleRegistration {
        public static void RegisterModules() {
            Type[] moduleTypes = {
                typeof(CommonModules.TimerModule),
                typeof(CommonModules.LogModule)
            };
            foreach (Type t in moduleTypes) {
                HttpApplication.RegisterModule(t);
            }
        }
    }
}
```

Применяемый в этом файле атрибут сборки `PreApplicationStartMethod` сообщает ASP.NET Framework о необходимости вызвать метод `RegisterModules()` класса `ModuleRegistration`, когда приложение запускается; указанный метод должен быть объявлен как `public` и `static`.

Внутри класса `RegisterModules` мы вызываем статический метод `HttpApplication.RegisterModule()` для регистрации созданных модулей. Это обеспечивает автоматическую настройку модулей в любом проекте ASP.NET Framework, к которому добавлена сборка `CommonModules`, причем без добавления элементов в файл `Web.config`.

Финальный шаг заключается в импортировании сборки, созданной проектом `CommonModules`, в проект `Events`. Щелкните правой кнопкой мыши на проекте `Events` в окне Solution Explorer и выберите в контекстном меню пункт Add Reference. Выберите категорию Solution (Решение) и найдите запись `CommonModules`. Отметьте флажок, как показано на рис. 14.2, и щелкните на кнопке OK, чтобы закрыть диалоговое окно и добавить ссылку на сборку.

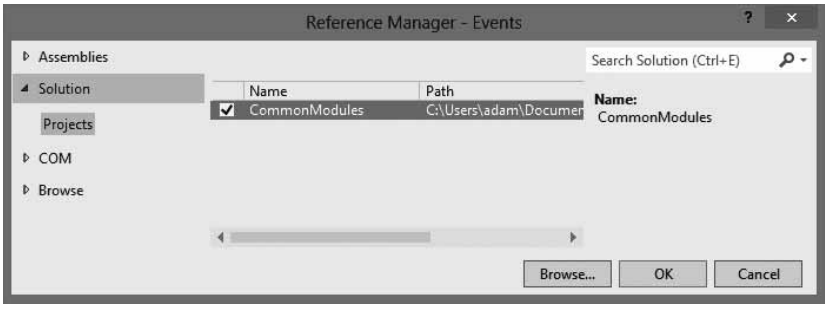

Рис. 14.2. Добавление в проект `Events` ссылки на сборку `CommonModules`

Для тестирования работоспособности модулей запустите приложение и перейдите на файлы веб-форм `Default.aspx` и `Params.aspx`. В окне Output среды Visual Studio вы увидите примерно такой вывод:

```
Request for /Default.aspx - code 200
Duration: /Default.aspx 16.0107ms
Request for /Params.aspx - code 500
Duration: /Params.aspx 2.0013ms
```

Работа с событиями модулей

Модули не должны существовать в изоляции, и мы можем избежать дублирования кода, открывая доступ к функциональности с применением событий — это позволяет создавать модули, основанные на возможностях других модулей. В последующих разделах будет показано, как добавлять событие к модулю, находить модуль и регистрировать обработчик для события.

Определение события модуля

Мы создадим новый модуль, который хранит детали о среднем времени, затраченном на обработку запроса. Это требует измерения длительности обработки каждого отдельного запроса — то, что мы не хотим делать в новом модуле, т.к. данная функциональность уже доступна в классе `TimerModule`. В листинге 14.9 демонстрируется добавление в файл `TimerModule.cs` проекта `CommonModules` события, которое позволит модулю публиковать свои данные о времени.

Листинг 14.9. Добавление события в файл `TimerModule.cs` проекта `CommonModules`

```
using System;
using System.Web;

namespace CommonModules {

   public class TimerEventArgs : EventArgs {
      public double Duration { get; set; }
   }

   public class TimerModule : IHttpModule {
      private DateTime startTime;
      public event EventHandler<TimerEventArgs> RequestTimed;

      public void Init(HttpApplication app) {
         app.BeginRequest += HandleEvent;
         app.EndRequest += HandleEvent;
      }

      private void HandleEvent(object src, EventArgs args) {
         HttpApplication app = src as HttpApplication;
         switch (app.Context.CurrentNotification) {
            case RequestNotification.BeginRequest:
               startTime = app.Context.Timestamp;
               break;
            case RequestNotification.EndRequest:
               double elapsed =
                  DateTime.Now.Subtract(startTime).TotalMilliseconds;
               System.Diagnostics.Debug.WriteLine(
                 string.Format("Duration: {0} {1}ms",
                               app.Request.RawUrl,
                               elapsed));
               if (RequestTimed != null) {
                 RequestTimed(this, new TimerEventArgs { Duration = elapsed });
               }
               break;
         }
      }

      public void Dispose() {
         // Ничего не делать.
      }
   }
}
```

Здесь определено событие по имени `RequestTimed`, отправляющее объект `TimerEventArgs` своим обработчикам. В классе `TimerEventArgs` определено свойство типа `double` по имени `Duration`, которое предоставляет доступ к информации, связанной со временем.

Обработка события модуля

Мы добавили в проект `Events` новый файл класса под названием `AverageTimeModule.cs` и определили в нем модуль, который будет отслеживать среднее время обработки запроса. Реализация этого модуля приведена в листинге 14.10.

Листинг 14.10. Класс `AverageTimeModule`

```
using System.Web;
using CommonModules;

namespace Events {
   public class AverageTimeModule : IHttpModule {
      private static double totalTime;
      private static int requestCount;
      private static object lockObject = new object();

      public void Init(HttpApplication app) {
         for (int i = 0; i < app.Modules.Count; i++ ) {
            if (app.Modules[i] is TimerModule) {
               (app.Modules[i] as TimerModule).RequestTimed += (src, args) => {
                  addNewDataPoint(args.Duration);
               };
               break;
            }
         }
      }
      private void addNewDataPoint(double duration) {
         lock (lockObject) {
            double ave = (totalTime += duration) / (++requestCount);
            System.Diagnostics.Debug.WriteLine(
               string.Format("Average request duration: {0:F2}ms", ave));
         }
      }
      public void Dispose() {
         // Ничего не делать.
      }
   }
}
```

В этом коротком файле класса предпринимаются два важных действия. Вспомните, что для обслуживания запросов среда ASP.NET Framework может создавать множество объектов `HttpApplication`, при этом каждый из них будет иметь экземпляр `TimerModule`, инициирующий события `RequestTimed`, и экземпляр `AverageTimeModule`, который обрабатывает эти события. Мы хотим собирать всю информацию, связанную со временем, а не только ту, что выдается единственным экземпляром `HttpApplication` и его модулями.

Мы обеспечиваем совместное использование всеми экземплярами класса `AverageTimeModule` одних и тех же значений данных за счет объявления переменных `totalTime` и `requestCount` как `static`. Мы хотим удостовериться, что не пытаемся обновлять указанные переменные одновременно из двух экземпляров обработчика, поэтому применяем в методе `addNewDataPoint()` оператор `lock` со ссылкой блокировки в виде статического объекта (который необходим, чтобы обеспечить использование одной и той же ссылки для блокировки всеми экземплярами класса модуля).

Внимание! Любой вид кода, который инициирует обработку запросов через блок `lock` или другой примитив синхронизации, будет значительно снижать производительность веб-приложения и не должен применяться в реальном проекте. Существуют приемы обеспечения целостности данных, не причиняя ущерб чему бы то ни было, однако они требуют подробного объяснения концепций параллельного программирования, которые не связаны напрямую с ASP.NET. За дополнительными сведениями обращайтесь к книге Адама Фримена *Pro .Net 4 Parallel Programming in C#* (Apress, 2010 г.).

Нахождение другого модуля

Обеспечение накопления всех данных является важным аспектом, но в настоящей главе мы больше всего заинтересованы в том, как один модуль может отыскать другой, чтобы можно было зарегистрировать обработчик событий. Модули могут быть обнаружены через свойство `Modules`, определенное в классе `HttpApplication`. Это свойство возвращает объект `HttpModulesCollection`, который представляет собой коллекцию реализаций `IHttpModule`, зарегистрированных в ASP.NET Framework. Свойства, которые определены в классе `HttpModulesCollection`, описаны в табл. 14.2.

Таблица 14.2. Свойства, определенные в классе `HttpModulesCollection`

Свойство	Описание
`AllKeys`	Возвращает строковый массив, содержащий имена всех модулей, которые были зарегистрированы
`Count`	Возвращает количество модулей, которые были зарегистрированы

В дополнение к этим свойствам класс `HttpModulesCollection` определяет индексаторы в стиле массивов, которые позволяют извлекать объекты `IHttpModule` из коллекции по имени или по индексу. Мы находим интересующие модули за счет инспектирования типа каждой реализации `IHttpModule`, содержащейся в `HttpModulesCollection`:

```
...
public void Init(HttpApplication app) {
    for (int i = 0; i < app.Modules.Count; i++ ) {
        if (app.Modules[i] is TimerModule) {
            (app.Modules[i] as TimerModule).RequestTimed += (src, args) => {
                addNewDataPoint(args.Duration);
            };
            break;
        }
    }
}
...
```

Каждый модуль проверяется по очереди, а для экземпляров `TimerModule` с помощью лямбда-выражения регистрируется обработчик события `RequestTimed`. Класс `HttpModulesCollection` в действительности предназначен для поиска модуля по имени, но в данном примере это не подходит по причинам, которые объясняются далее в главе.

Совет. Беспокоиться о порядке регистрации модулей не следует. Экземпляры всех модулей создаются до вызова метода `Init()`, так что каждый модуль сможет найти любой другой модуль при выполнении метода `Init()`.

Перед использованием нового модуля средой ASP.NET Framework он должен быть зарегистрирован. В листинге 14.11 показано добавление в файл `Web.config` проекта `Events`.

Листинг 14.11. Регистрация модуля `AverageTime` в файле `Web.config`

```
<?xml version="1.0"?>

<configuration>
    <system.web>
      <compilation debug="true" targetFramework="4.5" />
      <httpRuntime targetFramework="4.5" />
    </system.web>
```

```
  <system.webServer>
    <modules>
      <add name="ParamsProtection" type="Events.ParamsModule"/>
      <add name="AverageTime" type="Events.AverageTimeModule"/>
    </modules>
  </system.webServer>
</configuration>
```

Благодаря такому добавлению, мы получаем в окне Output среды Visual Studio скользящее среднее значение для времени, затраченного на обработку запроса:

```
Request for /ListModules.aspx - code 200
Duration: /ListModules.aspx 273.182ms
Average request duration: 273.18ms
Request for /Params.aspx - code 500
Duration: /Params.aspx 2.0014ms
Average request duration: 137.59ms
```

Нахождение модулей по имени

В предыдущем примере поиск нужного модуля осуществлялся путем просмотра всех зарегистрированных модулей и проверки их типов. Существует более простой способ — модуль можно находить по имени, под которым он был зарегистрирован. Чтобы показать, как это работает (и подготовиться к демонстрации ряда других средств), мы добавили в класс `AverageTimeModule` событие (листинг 14.12).

Листинг 14.12. Добавление события в класс `AverageTimeModule` из файла `AverageTimeModule.cs`

```
using System;
using System.Web;
using CommonModules;

namespace Events {

   public class AverageTimeEventArgs : EventArgs {
      public double AverageTime { get; set; }
   }

   public class AverageTimeModule : IHttpModule {
      private static double totalTime;
      private static int requestCount;
      private static object lockObject = new object();
      public event EventHandler<AverageTimeEventArgs> NewAverage;

      public void Init(HttpApplication app) {
         for (int i = 0; i < app.Modules.Count; i++ ) {
            if (app.Modules[i] is TimerModule) {
               (app.Modules[i] as TimerModule).RequestTimed += (src, args) =>
{
                  addNewDataPoint(args.Duration);
               };
               break;
            }
         }
      }

      private void addNewDataPoint(double duration) {
         lock (lockObject) {
            double ave = (totalTime += duration) / (++requestCount);
```

```
                System.Diagnostics.Debug.WriteLine(
                    string.Format("Average request duration: {0:F2}ms", ave));
                if (NewAverage != null) {
                  NewAverage(this, new AverageTimeEventArgs { AverageTime = ave });
                }
            }
        }
        public void Dispose() {
            // Ничего не делать.
        }
    }
}
```

Мы определили событие по имени `NewAverage`, которое отправляет обработчикам объект `AverageTimeEventArgs`, содержащий последние данные. В листинге 14.13 приведен модифицированное содержимое файла `Global.asax.cs`, в котором производится поиск модуля в методе `Init()` и установка обработчика для нового события. Для нахождения модуля, зарегистрированного под именем `AverageTime`, применяется индексатор в стиле массива.

Листинг 14.13. Обработка события модуля в файле `Global.asax.cs`

```
using System;
using System.Web;

namespace Events {
    public class Global : System.Web.HttpApplication {

        protected void Application_Start(object sender, EventArgs e) {
            EventCollection.Add(EventSource.Application, "Start");
            Application["message"] = "Application Events";
        }
        protected void Application_End(object sender, EventArgs e) {
            EventCollection.Add(EventSource.Application, "End");
        }
        public override void Init() {
            IHttpModule mod = Modules["AverageTime"];
            if (mod is AverageTimeModule) {
                ((AverageTimeModule)mod).NewAverage += (src, args) => {
                    Response.Write(string.Format("<h3>Ave time: {0:F2}ms</h3>",
                        args.AverageTime));
                };
            }
        }
    }
}
```

На заметку! Мы не можем воспользоваться этим приемом для поиска модулей в проекте `CommonModules`, потому что при автоматической регистрации модуля ASP.NET Framework создает чрезвычайно длинное имя — позже в главе будет представлен пример такого имени. Автоматически зарегистрированные модули необходимо искать по типу, как было показано ранее.

Мы по-прежнему проверяем тип полученного объекта — он может быть `null`, т.е. модуль с таким именем не существует, или иметь тип, отличный от ожидаемого, предполагая, что код не согласован с файлом `Web.config`. Если тип объекта является ожидаемым, мы применяем лямбда-выражение для обработки события.

Совет. Метод `HttpApplication.Init()` вызывается после того, как все объекты модулей созданы и все их методы `Init()` выполнены, что предоставляет великолепную возможность установки обработчиков событий — модули готовы к обработке событий жизненного цикла запросов, а событие `BeginEvent` еще не было отправлено. Удостоверьтесь, что в реализации метода `Init()` присутствует ключевое слово `override`, иначе этот код не будет вызван.

Код, добавленный в глобальный класс приложения, будет вставлять в ответ для каждого запроса элемент `h3` со средним временем обработки, как показано на рис. 14.3. Строки в таблице отсутствуют, т.к. в этой главе мы не используем класс `EventCollection` для записи событий жизненного цикла.

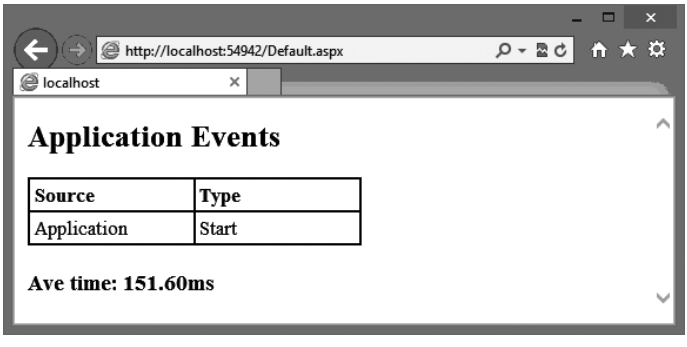

Рис. 14.3. Добавление среднего времени обработки запроса в каждый ответ

Тот же самый прием можно применять внутри модуля. Мы используем глобальный класс приложения, поскольку хотим продемонстрировать альтернативный подход, который *не может* быть применен в модуле. В листинге 14.14 определен декларативный обработчик для события `NewAverage`, которое было определено в классе `AverageTimeModule`.

Листинг 14.14. Определение декларативного обработчика для события модуля

```
using System;
using System.Web;

namespace Events {
    public class Global : System.Web.HttpApplication {

        protected void Application_Start(object sender, EventArgs e) {
            EventCollection.Add(EventSource.Application, "Start");
            Application["message"] = "Application Events";
        }

        protected void Application_End(object sender, EventArgs e) {
            EventCollection.Add(EventSource.Application, "End");
        }

        public void AverageTime_NewAverage(object src,
                                           AverageTimeEventArgs args) {
            Response.Write(string.Format("<h3>Ave time: {0:F2}ms</h3>",
                args.AverageTime));
        }
    }
}
```

Декларативный обработчик события создается точно так же, как для событий жизненного цикла, за исключением того, что имя обработчика образовано конкатенацией значения атрибута `name`, указанного при регистрации модуля в файле `Web.config`, символа подчеркивания и имени события.

Работа со встроенными модулями

В начале главы 13 мы создали новый глобальный класс приложения и указали, что стандартный код содержит методы трех категорий. В качестве напоминания ниже представлен код, который создает Visual Studio:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.SessionState;

namespace Events {
    public class Global : System.Web.HttpApplication {

        protected void Application_Start(object sender, EventArgs e) {}

        protected void Session_Start(object sender, EventArgs e) {}

        protected void Application_BeginRequest(object sender, EventArgs e) {}

        protected void Application_AuthenticateRequest(object sender, EventArgs e) {}

        protected void Application_Error(object sender, EventArgs e) {}

        protected void Session_End(object sender, EventArgs e) {}

        protected void Application_End(object sender, EventArgs e) {}
    }
}
```

Теперь, когда вы знаете, как создавать декларативные обработчики для событий, определяемых модулями, должно стать ясно, что два метода, которые не были объяснены в главе 13 — `Session_Start()` и `Session_End()` — являются обработчиками для событий `Start` и `End`, которые определены в модуле, зарегистрированном под именем `Session`.

Инфраструктура ASP.NET Framework содержит набор модулей, разработанных в Microsoft, которые предоставляют функциональность, помогающую обрабатывать запросы. Получить детальные сведения об этих модулях можно с помощью свойства `HttpApplication.Modules`, и для этого мы добавили в проект новую веб-форму под названием `ListModules.aspx` (листинг 14.15).

Листинг 14.15. Содержимое файла отделенного кода `ListModules.aspx.cs`

```
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Events {

    public class ModuleDescription {
        public string Name { get; set; }
        public string TypeName { get; set; }
    }

    public partial class ListModules : System.Web.UI.Page {
```

```
        public IEnumerable<ModuleDescription> GetModules() {
            HttpModuleCollection modules = Context.ApplicationInstance.Modules;
            foreach (string key in modules.AllKeys.OrderBy(x => x)) {
                yield return new ModuleDescription {
                    Name = key,
                    TypeName = modules[key].GetType().ToString()
                };
            }
        }
    }
}
```

В классе отделенного кода `ListModules` определен метод `ListModules()`, который будет вызываться внутри фрагмента кода веб-формы. Целью этого метода является генерация коллекции объектов, описывающих модули, которые были зарегистрированы в объекте `HttpApplication`.

Для получения экземпляра `HttpApplication` внутри класса отделенного кода веб-формы применяется свойство `Context.ApplicationInstance`. При наличии объекта `HttpApplication` мы обращается к свойству `Modules`, чтобы получить объект `HttpModulesCollection`, представляющий собой простую коллекцию с именами классов модулей, под которыми они были зарегистрированы (другими словами, значения атрибутов `name` элементов `add` внутри элемента `modules` файла `Web.config`).

В листинге 14.15 с помощью свойства `AllKeys` извлекается набор имен, который используется (посредством LINQ) для генерации последовательности объектов `ModuleDescription`, возвращаемой в качестве результата из метода `GetModules()`. Каждый объект `ModuleDescription` содержит имя, под которым модуль был зарегистрирован, и тип класса реализации интерфейса `IHttpModule`. В листинге 14.16 приведено содержимое файла `ListModules.aspx`, в котором с помощью элемента управления `Repeater` отображаются детали, связанные с модулями.

Листинг 14.16. Содержимое файла `ListModules.aspx`

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="ListModules.aspx.cs"
    Inherits="Events.ListModules" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>
    <style>
        th, td { border-bottom: thin solid black; text-align: left;
            padding: 3px;}
        td span { display: inline-block; text-overflow: ellipsis;
            overflow: hidden; white-space:nowrap; width: 300px;}
        table { border-collapse: collapse;}
    </style>
</head>
<body>
    <div>
        <table>
            <tr><th>Name</th><th>Type</th></tr>
            <asp:Repeater ID="Repeater1" ItemType="Events.ModuleDescription"
                SelectMethod="GetModules" runat="server">
                <ItemTemplate>
```

```
                <tr>
                    <td><span><%#: Item.Name %></span></td>
                    <td><%#: Item.TypeName %></td>
                </tr>
            </ItemTemplate>
        </asp:Repeater>
    </table>
  </div>
</body>
</html>
```

Запустив приложение и перейдя на веб-форму `ListModules.aspx`, вы увидите список зарегистрированных модулей, как показано на рис. 14.4.

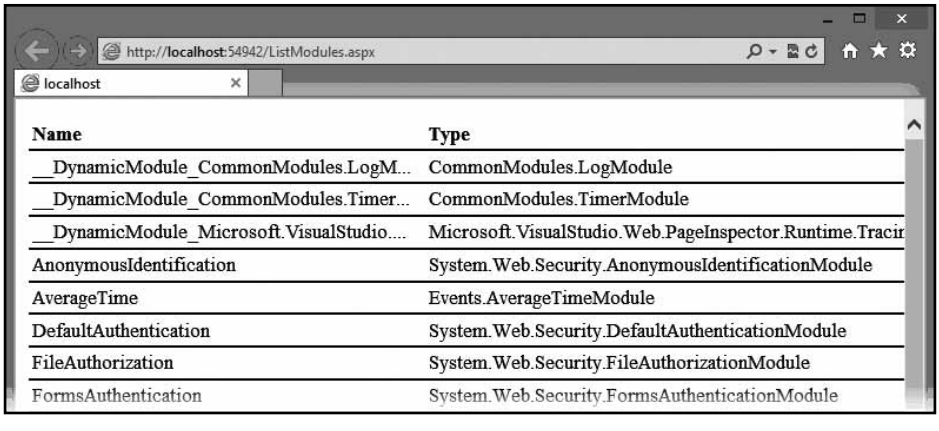

Рис. 14.4. Отображение списка модулей, зарегистрированных в приложении ASP.NET Framework

Для каждого модуля веб-форма `ListModules.aspx` отображает имя и тип. Имена первых трех модулей были сокращены из-за их длины — это имена, сгенерированные ASP.NET Framework для автоматически регистрируемых модулей. Например, вот как выглядит полное имя, сгенерированное для класса `LogModule`:

```
__DynamicModule_CommonModules.LogModule, CommonModules, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=null_602f9111-0495-4382-917f-90d4ffb250d4
```

Это имя содержит детали, связанные со сборкой, в которой находится класс модуля, что обеспечивает уникальность идентификации классов, но делает невозможным их применение при поиске модуля по имени (по этой причине в проекте `CommonModules` было показано, как искать модуль по типу). Два длинных имени относятся к классам `CommonModules`, а третье поддерживает средство Page Inspector (Инспектор страниц) среды Visual Studio, описанное в главе 5.

В настоящий момент нас не заботят эти три модуля — мы заинтересованы в остальных. В табл. 14.3 перечислены все встроенные модули и приведена информация об их классах, назначении и определяемых событиях.

Таблица 14.3. Модули в приложении ASP.NET Framework

Модуль	Описание
AnonymousIdentification	Этот модуль реализован классом System.Web.Security.AnonymousIdentificationModule и отвечает за уникальную идентификацию запросов, что позволяет применять средства, подобные пользовательским профилям (глава 18), даже если пользователь не аутентифицирован. Определяет событие Creating, которое предоставляет возможность переопределения идентификации. Событие Creating отправляет обработчикам событий экземпляр класса AnonymousIdentificationEventArgs
DefaultAuthentication	Этот модуль реализован классом System.Web.Security.DefaultAuthenticationModule и отвечает за то, что в свойстве User объекта HttpContext устанавливается объект, реализующий интерфейс IPrincipal, если это не было сделано каким-то другим модулем аутентификации. Класс HttpContext объяснялся в главе 13, а интерфейс IPrincipal будет описан в главах 25 и 26. Модуль определяет событие Authenticate, которое инициируется, когда модуль устанавливает свойство User, и отправляет обработчикам событий экземпляр класса DefaultAuthenticationEventArgs
FileAuthorization	Этот модуль реализован классом System.Web.Security.FileAuthorizationModule и отвечает за то, что пользователь имеет доступ к файлу, связанному с запросом, когда применяется аутентификация Windows. Аутентификация ASP.NET описана в главе 25, но интеграция с Windows в этой книге не рассматривается
FormsAuthentication	Этот модуль реализован классом System.Web.Security.FormsAuthenticationModule и устанавливает значение свойства HttpContext.User при использовании аутентификации с помощью форм. Аутентификация с помощью форм объясняется в главе 25. Модуль определяет событие Authenticate, которое позволяет переопределять значение свойства User и отправляет обработчикам событий объект FormsAuthenticationEventArgs
OutputCache	Этот модуль реализован классом System.Web.Caching.OutputCacheModule и отвечает за кеширование ответов, отправляемых браузеру. Работа средств кеширования вывода описана в главе 20. Модуль не определяет никаких событий
Profile	Этот модуль реализован классом System.Web.Profile.ProfileModule и отвечает за ассоциирование данных пользовательского профиля с запросом. (Данные профиля подробно рассматриваются в главе 18.) Событие MigrateAnonymous инициируется, когда анонимный пользователь заходит в приложение, и отправляет обработчикам событий объект ProfileMigrateEventArgs. Событие Personalize возникает, когда данные профиля ассоциируются с запросом, и предоставляет возможность переопределения используемых данных (обработчикам отправляется объект ProfileEventArgs)
RoleManager	Этот модуль реализован классом System.Web.Security.RoleManagerModule и отвечает за назначение запросу ролей, которые имеет пользователь. Роли описаны в главе 25. Модуль определяет событие GetRoles, которое позволяет переопределять информацию о ролях, связанную с запросом. Обработчикам событий отправляется объект RoleManagerEventArgs

Окончание табл. 14.3

Модуль	Описание
ScriptModule-4.0	Этот модуль реализован классом System.Web.Handlers.ScriptModule и отвечает за поддержку запросов Ajax, которые объясняются в части IV. Модуль не определяет никаких событий
ServiceModel-4.0	Этот модуль реализован классом System.ServiceModel.Activation.ServiceHttpModule. Модуль применяется для активизации веб-служб ASP.NET; модель веб-служб в книге не рассматривается, т.к. мы предпочитаем пользоваться новым средством Web API, которое описано в части IV
Session	Этот модуль реализован классом System.Web.SessionState.SessionStateModule и отвечает за ассоциирование данных сеанса с запросом. Событие Start инициируется при запуске нового сеанса, а событие End — когда сеанс завершает работу. Оба события отправляют обработчикам стандартный объект EventArgs
UrlAuthorization	Этот модуль реализован классом System.Web.Security.UrlAuthorizationModule и обеспечивает пользователям авторизацию для доступа к запрашиваемым веб-формам. Система авторизации рассматривается в главе 25. Модуль не определяет никаких событий
UrlMappingsModule	Этот модуль реализован классом System.Web.UrlMappingsModule и отвечает за реализацию средства отображения URL, которое описано в главе 22. Модуль не определяет никаких событий
UrlRoutingModule-4.0	Этот модуль реализован классом System.Web.Routing.UrlRoutingModule и отвечает за реализацию средства маршрутизации URL, которое описано в главах 23 и 24. Модуль не определяет никаких событий
WindowsAuthentication	Этот модуль реализован классом System.Web.Security.WindowsAuthenticationModule и отвечает за установку значения свойства HttpContext.User, когда применяется аутентификация Windows. Модуль определяет событие Authenticate, которое позволяет переопределить удостоверение, ассоциированное с запросом. Обработчикам событий отправляется объект WindowsAuthenticationEventArgs

Совет. Рекомендуется самостоятельно запустить этот пример, потому что последние обновления ASP.NET Framework могут привести к получению набора модулей, который отличается от рассмотренного здесь перечня.

Вернувшись к глобальному классу приложений, можно заметить, что модуль Session реализован классом SessionStateModule, который определяет события Start и End, отражающие установку и уничтожение состояния сеанса для запроса. Средство состояния сеанса подробно объясняется в главе 18.

Собираем все вместе

В завершение этой главы мы создадим более сложный модуль, чтобы продемонстрировать совместное применение всех показанных здесь приемов. Преимущество модулей связано с тем, что они позволяют вставлять специальную логику в любую точку процесса обработки запросов. С модулями не связано ничего такого, чего нельзя было

бы сделать в других местах ASP.NET Framework, но нам нравится автономность и возможность многократного использования модулей, и мы предпочитаем применять их для реализации сквозной функциональности в разрабатываемом приложении.

В этом разделе мы планируем создать модуль, который устанавливает культуру на основе информации, предоставленной в запросе. Это приведет к переопределению стандартной культуры сервера приложений и корректной установке форматов для денежных значений и дат (помимо прочих настроек).

Итак, добавим в папку `Events` новый файл веб-формы по имени `Price.aspx`. Содержимое этого файла представлено в листинге 14.17.

Листинг 14.17. Содержимое файла `Price.aspx`

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Price.aspx.cs"
         Inherits="Events.Price" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
</head>
<body>
   <p>Today's date is <%= DateTime.Now.ToShortDateString() %></p>
   <p>A new shirt costs <%= 20.ToString("C") %></p>
</body>
</html>
```

Для тестового сервера, настроенного для США, запрос веб-формы `Price.aspx` даст в результате следующий вывод:

```
Today's date is 1/8/2014
A new shirt costs $20.00
```

На тестовом сервере, который настроен для Соединенного Королевства, применяются другие форматы для денежных значений и дат. Наша цель заключается в обнаружении информации о локали, предоставляемой браузером, и форматировании данных в соответствие с ней. Для этого мы создаем в папке `Events` новый файл класса по имени `LocaleModule.cs`. В листинге 14.18 показано определение модуля, который решает проблему, связанную с локалью.

Листинг 14.18. Содержимое файла `LocaleModule.cs`

```
using System;
using System.Globalization;
using System.Threading;
using System.Web;

namespace Events {
   public class LocaleModule : IHttpModule {
      public void Init(HttpApplication app) {
         app.BeginRequest += HandleEvent;
      }
      protected void HandleEvent(object src, EventArgs args) {
         string[] langs = ((HttpApplication)src).Request.UserLanguages;
         if (langs != null && langs.Length > 0 && langs[0] != null) {
            try {
               Thread.CurrentThread.CurrentCulture =
                     new CultureInfo(langs[0]);
               //Thread.CurrentThread.CurrentCulture = new CultureInfo("en-GB");
```

```
                } catch {}
            }
        }
        public void Dispose() {
        }
    }
}
```

Этот модуль обрабатывает событие `BeginRequest` и использует свойство `HttpRequest.UserLanguages` для получения набора языков, указанных браузером. Запросы могут содержать информацию о нескольких языках, которые пользователь желает принимать, и свойство `UserLanguages` возвращает языки в порядке предпочтения. В модуле применен очень простой подход — выбирается первый указанный язык и предпринимается попытка использовать его при установке локали для запроса.

В листинге 14.19 показано, как зарегистрировать этот модуль в файле `Web.config`.

Листинг 14.19. Регистрация модуля `LocaleModule` в файле `Web.config`

```
<?xml version="1.0"?>
<configuration>
    <system.web>
      <compilation debug="true" targetFramework="4.5" />
      <httpRuntime targetFramework="4.5" />
      <globalization culture="en-US" uiCulture="en-US"/>
    </system.web>
  <system.webServer>
    <modules>
      <add name="ParamsProtection" type="Events.ParamsModule"/>
      <add name="AverageTime" type="Events.AverageTimeModule"/>
      <add name="Locale" type="Events.LocaleModule"/>
    </modules>
  </system.webServer>
</configuration>
```

Чтобы увидеть эффект от применения модуля `LocaleModule`, необходимо запустить приложение и запросить веб-форму `Price.aspx` — однако может понадобиться изменить локаль в настройках операционной системы и браузера. Если по какой-либо причине вы не хотите изменять локаль, удалите символы комментария со строки кода в листинге 14.18, которая эмулирует запрос, указывающий культуру `en-GB`. Ниже приведен вывод, полученный в результате запроса из браузера с установленной локалью `en-GB`, который отображает дату в корректном формате и символ валюты, как принято в Соединенном Королевстве:

```
Today's date is 08/01/2014
A new shirt costs £20.00
```

Резюме

В этой главе было показано, как применять модули для переноса функциональности из глобального класса приложения в автономный и многократно используемый класс. Мы продемонстрировали создание модулей внутри проекта ASP.NET и в отдельном проекте, а также различные способы регистрации модулей. Мы объяснили, как модули могут инициировать события, и описали разные подходы к нахождению модулей для регистрации методов обработчиков. В конце главы мы создали модуль, который устанавливает информацию о локали для запроса на основе данных из HTTP-заголовков, отправленных браузером. В главе 15 будет рассмотрен еще один способ настройки процесса обработки запросов — *обработчики*.